Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Первоначальная сборка системы (дерева иерархии объектов, модели системы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Поиск головного объекта выполняется от последнего созданного объекта. Первоначально последним созданным объектом считается корневой объект. Если для головного объекта обнаруживается дубляж имени в непосредственно подчиненных объектах, то объект не создается. Если обнаруживается дубляж имени на дереве иерархии объектов, то объект не создается. Если номер класса объекта задан некорректно, то объект не создается.

Вывод иерархического дерева объектов на консоль

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных моделях систем динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны. Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Расширить функциональность базового класса:

- метод поиска объекта на ветке дереве иерархии от текущего по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на искомой ветке дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод вывода иерархии объектов (дерева или ветки) от текущего объекта;
- метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта;
- метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется. При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:

1. Построить дерево объектов системы (в методе корневого объекта построения исходного дерева объектов).

2. В методе корневого объекта запуска моделируемой системы реализовать:

2.1 Вывод на консоль иерархического дерева объектов в следующем виде:

```
root
    ob_1
        ob_2
    ob_3
        ob_4
            ob_5
        ob_6
            ob_7
```

где: root - наименование корневого объекта (приложения).

2.2. Переключение готовности объектов согласно входным данным (командам).

2.3. Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:

```
root  is ready
    ob_1  is ready
        ob_2  is ready
    ob_3  is ready
        ob_4 is not ready
            ob_5 is not ready
        ob_6 is ready
            ob_7 is not ready
```

1.1 Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Последовательность ввода организовано так, что головной объект для очередного вводимого объекта уже присутствует на дереве иерархии объектов.

Первая строка

«Наименование корневого объекта»

Со второй строки

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

. . . . .

endtree

Со следующей строки вводятся команды включения или отключения объектов

«Наименование объекта» «Номер состояния объекта»

Пример ввода

```
app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
object_02 object_05 5
object_01 object_07 2
endtree
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
```

1.2 Описание выходных данных

Вывести иерархию объектов в следующем виде:

```
Object tree
«Наименование корневого объекта»
    «Наименование объекта 1»
        «Наименование объекта 2»
    «Наименование объекта 3»
. . . . .
The tree of objects and their readiness
«Наименование корневого объекта» «Отметка готовности»
    «Наименование объекта 1» «Отметка готовности»
        «Наименование объекта 2» «Отметка готовности»
    «Наименование объекта 3» «Отметка готовности»
. . . . .
«Отметка готовности» - равно «is ready» или «is not ready»
```

Отступ каждого уровня иерархии 4 позиции.

Пример вывода

```
Object tree
app_root
    object_01
        object_07
    object_02
        object_04
        object_05
The tree of objects and their readiness
app_root is ready
    object_01 is ready
        object_07 is not ready
    object_02 is ready
        object_04 is ready
        object_05 is not ready
```

2 МЕТОД РЕШЕНИЯ

Объект стандартного потока ввода и вывода (cin, cout)

Условный оператор if

Оператор со счетчиком for

Оператор цикла с предусловием while

Для решения задачи было добавлено\изменено:

Класс cl_base:

- Поля:
  - state - тип int, модификатор доступа(private)
- Методы:
  - int obj_count(string s_name) - метод поиска объекта на ветке деревиерархии от текущего по имени, модификатор доступа(public)
  - cl_base* search_branchstring s_name) - метод поискаэлемента по имени от текущего, модификатор доступа(public)
  - cl_base* search_object(string s_name) - метод поиска объекта на дереве иерархии по имени, модификатор доступа(public)
  - cl_base* search_tree(string s_name) - метод поиска объекта на дереве иерархии по имени, модификатор доступа(public)
  - void set_status(int state) - установка состояния объекта, модификатор доступа(public)
  - bool get_status() - возврат значение состояния объекта, модификатор доступа(public)
  - void print_tree(int spaces = 4) - метод вывода иерархии объектов (дерева или ветки) от текущего объекта, модификатор доступа(public)
  - void print_status(int spaces = 4) - метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта,

модификатор доступа(public)

Класс cl_2:

- Методы:
  - cl_2(cl_base *p_head, string s_name) - параметризированный конструктор, модификатор доступа(public)

Класс cl_3:

- Методы:
  - cl_3(cl_base *p_head, string s_name) - параметризированный конструктор, модификатор доступа(public)

Класс cl_4:

- Методы:
  - cl_4(cl_base *p_head, string s_name) - параметризированный конструктор, модификатор доступа(public)

Класс cl_5:

- Методы:
  - cl_5(cl_base *p_head, string s_name) - параметризированный конструктор, модификатор доступа(public)

Класс cl_6:

- Методы:
  - cl_6(cl_base *p_head, string s_name) - параметризированный конструктор, модификатор доступа(public)

- Таблица 1 – Иерархия наследования классов

№	ИМЯ КЛАССА	КЛАСС НАСЛЕДНИК	МОДИФИКАТОР ДОСТУПА ПРИ НАСЛЕДОВАНИИ	ОПИСАНИЕ	НОМЕР	КОММЕНТАРИЙ
1	CL_BASE			БАЗОВЫЙ КЛАСС		
		CL_APPLICATION	PUBLIC		2	
		CL_1	PUBLIC		3	
		CL_2	PUBLIC		4	
		CL_3	PUBLIC		5	
		CL_4	PUBLIC		6	
		CL_5	PUBLIC		7	
		CL_6	PUBLIC		8	
2	CL_APPLICATION			ПРОИЗВОДНЫЙ КЛАСС ОТ КЛАССА CL_BASE		
3	CL_1			ПРОИЗВОДНЫЙ КЛАСС ОТ КЛАССА С		

				L_BASE		
4	CL_2			ПРОИЗВОДНЫЙ КЛАСС ОТ КЛАССА CL_BASE		
5	CL_3			ПРОИЗВОДНЫЙ КЛАСС ОТ КЛАССА CL_BASE		
6	CL_4			ПРОИЗВОДНЫЙ КЛАСС ОТ КЛАССА CL_BASE		
7	CL_5			ПРОИЗВОДНЫЙ КЛАСС ОТ КЛАССА CL_BASE		
8	CL_6			ПРОИЗВОДНЫЙ КЛАСС ОТ КЛАССА CL_BASE		

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм метода obj_count класса cl_base

Функционал: Поиск объекта на ветке дереве иерархии от текущего поимени.

Параметры: string s_name.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода obj_count класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация поля counter = 0 типа int	2
2	get_name() == s_name	counter++	3
			3
3	p_sub_object имеется в списке p_sub_objects	Прибавление к переменной counter значение выполнения метода count с аргументов значения s_name объекта p_sub_objet	3
		Возрат значения поля counter	∅

3.2 Алгоритм метода search_object класса cl_base

Функционал: Поиск объекта на ветке иерархии по имени.

Параметры: string s_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода search_object класса cl_base

№	Предикат	Действия	№ перехода
1	obj_count(s_name) != 1	Возват нулевого указаеля nullpr	∅
			2
2	get_name() == s_name	Возврат указателя на текущий объект	∅
			3
3	p_sub_object имеется в списке p_sub_objects	Инициализация объекта p_found типа указателя на класс cl_base, путем вызова метода search_object с аргументом значения поля s_name объекта p_sub_object	4
			5
4	p_found != nullptr	Возврат указателя p_found	∅
			3
5		Возврат нулевого указателя nullptr	∅

3.3 Алгоритм метода search_branch класса cl_base

Функционал: Поиск элемента по имени от текущего.

Параметры: string s_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода search_branch класса cl_base

№	Предикат	Действия	№ перехода
1	count(s_name) != 1	Возврат нулевого указателя nullptr	∅
			2
2		Возврат работы метода search_object с аргументом значения поля s_name	∅

3.4 Алгоритм метода search_tree класса cl_base

Функционал: Поиска объекта на дереве иерархии по имени.

Параметры: string s_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода search_tree класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация переменной obj универсального типа указателем на текущий объект	2
2	obj -> get_head_object()	Присвоение переменной obj значение родительского объекта obj	2
			3
3	obj_count(s_name) != 1	Возрат нулевого указателя nullptr	∅
			4
4		Возврат работы метода search_object с аргументом значения поля s_name	∅

№	Предикат	Действия	№ перехода

3.5 Алгоритм метода get_status класса cl_base

Функционал: Возврат значение состояния объекта.

Параметры: нет.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_status класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения поля state	∅

3.6 Алгоритм метода set_status класса cl_base

Функционал: Установка состояния объекта.

Параметры: int state.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода set_status класса cl_base

№	Предикат	Действия	№ перехода
1	get_head_object() && ! get_head_object() -> GetObjectState()	Присвоение полю state значение 0	2
		Присвоение полю state значение параметра state	2

№	Предикат	Действия	№ перехода
2	!state		3
			∅
3	p_sub_object имеется в списке p_sub_objects	Вызов метода SetObjectState с аргументов значения поля state объекта p_sub_object	3
			∅

3.7 Алгоритм метода print_status класса cl_base

Функционал: Вывод иерархии объектов (дерева или ветки) и отметок ихготовности от текущего объекта.

Параметры: int spaces.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода print_status класса cl_base

№	Предикат	Действия	№ перехода
1		Вызов метода get_name для текущего объекта	2
2	get_status	Вывод " is ready"	3
		Вывод " is not ready"	3
3	!p_sub_objects.empty()		4
			∅
4	p_sub_object имеется в списке p_sub_objects	Вывод "\n"	5
			∅

№	Предикат	Действия	№ перехода
5		Инициализация переменной i = 0 типа int	6
6	i < spaces	Вывод " "	7
			8
7		i++	6
8		Вызов метода print_Status с аргументом spaces + 4 объекта p_sub_object	4

3.8 Алгоритм метода print_tree класса cl_base

Функционал: Вывод иерархии объектов (дерева или ветки) от текущего объекта.

Параметры: int spaces.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод метода get_name для текущего объекта	2
2	!p_sub_objects.empty()		3
			∅
3	p_sub_object имеется в списке p_sub_objects	Вывод "\n"	4

№	Предикат	Действия	№ перехода
			∅
4		Инициализация переменной i = 0 типа int	5
5	i < spaces	Вывод " "	6
			7
6		i++	5
7		Вызов метода print_status с аргументом spaces + 4 объекта p_sub_object	3

3.9 Алгоритм метода build_tree_objects класса cl_application

Функционал: Построение исходного древа иерархии.

Параметры: нет .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление полей s_head_name и s_sub_name типа string	2
2		Присвоение указателю p_head значение this	3
3		Присвоение указателю p_sub значение nullptr	4
4		Объявление полей classNum и state типа int	5

№	Предикат	Действия	№ перехода
5		Ввод значения поля s_head_name	6
6		Вызов метода set_name с аргументом s_head_name	7
7	true	Ввод значения поля s_head_name	8
			17
8	s_head_name == "endtree"	Выход из цикла	17
			9
9		Инициализация переменной указателя p_head на класс cl_base, путем вызова метода search_tree с аргументом значения s_head_name	10
10		Ввод значений полей s_sub_name и classNum	11
11	p_head && !p_head -> get_sub_object(s_sub_name)		12
			7
12	class_num = 2	Создание нового объекта класса cl_2 с аргументами в виде p_head, s_sub_name	7
			13
13	class_num = 3	Создание нового объекта класса cl_3 с аргументами в виде p_head, s_sub_name	7
			14
1	class_num = 4	Создание нового объекта класса cl_4 с аргументами	7

№	Предикат	Действия	№ перехода
4		в виде p_head, s_sub_name	
			15
15	class_num = 5	Создание нового объекта класса cl_5 с аргументами в виде p_head, s_sub_name	7
			16
16	class_num = 6	Создание нового объекта класса cl_6 с аргументами в виде p_head, s_sub_name	7
			7
17	cin >> s_head_name	Ввод значения поля state	18
			∅
18		Инициализация переменной obj универсального типа, путем вызова метода search_tree с аргументом значения s_head_name	19
19	obj	Вызов метода set_status объекта obj с аргументом значения state	17
			17

3.10 Алгоритм метода exec_app класса cl_application

Функционал: Используется для запуска приложения.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Object tree\n"	2
2		Вызов метода Print_tree	3
3		Вывод "\nThe tree of objects and their readiness\n"	4
4		Вызов метода Print_status	5
5		Возврат значения 0	∅

3.11 Алгоритм конструктора класса cl_2

Функционал: Вызов конструктора класса cl_base, алгоритмы классов cl_3, cl_4, cl_5, cl_6 анологичны класса cl_2.

Параметры: cl_base *p_head, string s_name.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base	∅

3.12 Алгоритм функции main

Функционал: Основная программа.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 13.

Таблица 13 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание корневого объекта приложения	2
2		Конструирование системы	3
3		Запуск программы	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-10.

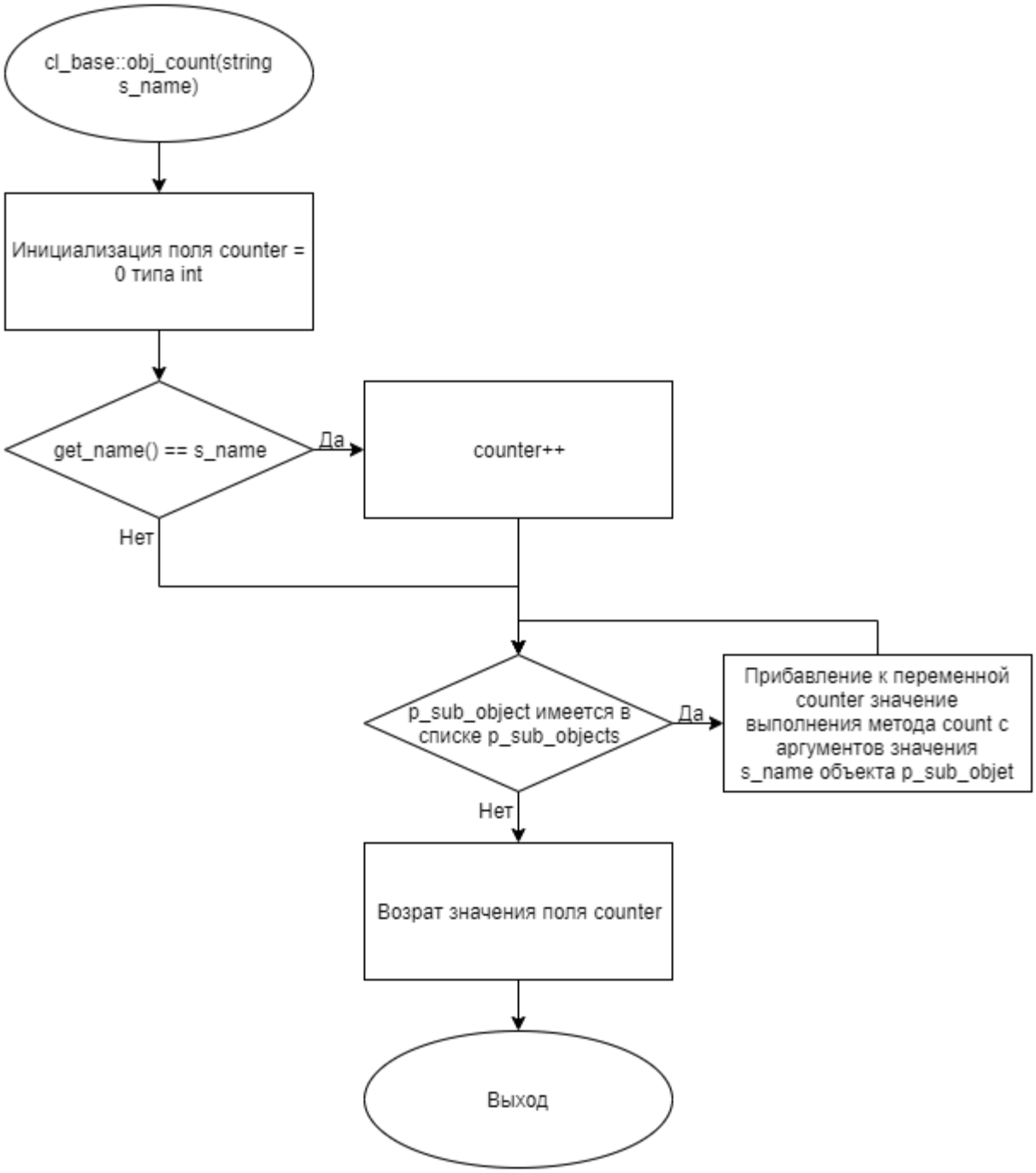

Рисунок 1 – Блок-схема алгоритма

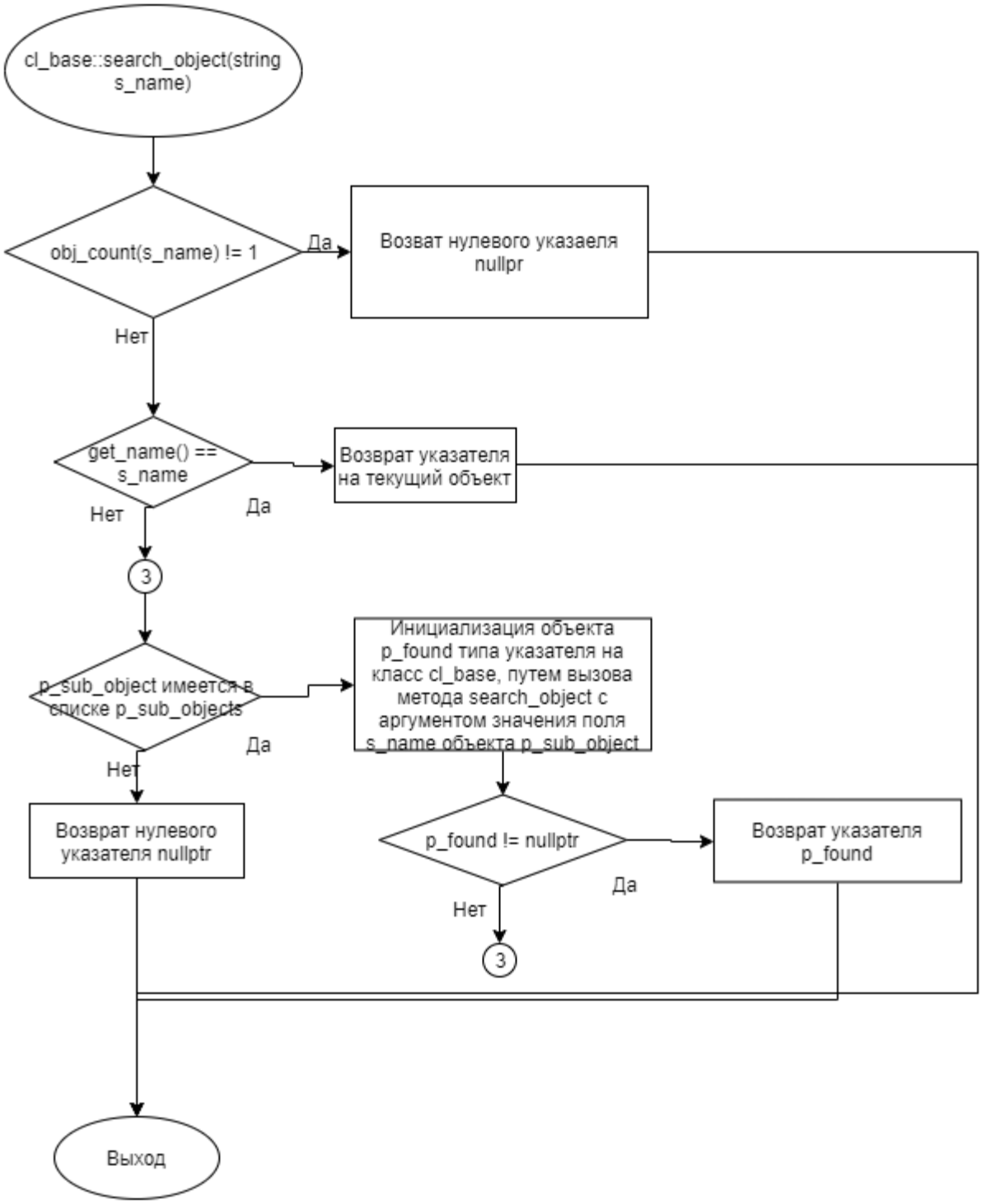

Рисунок 2 – Блок-схема алгоритма

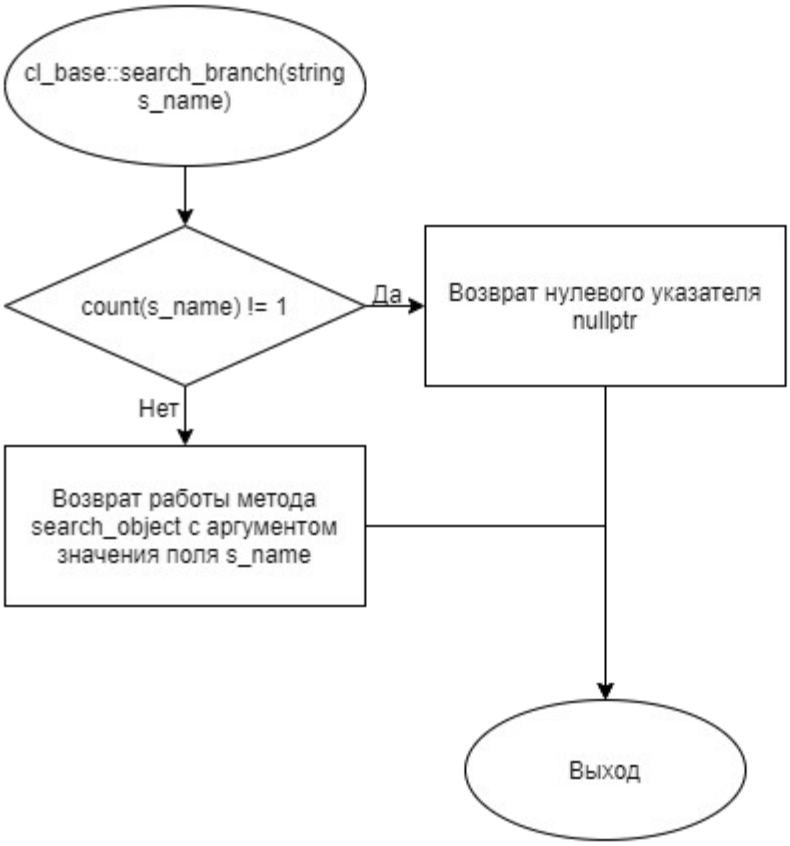

Рисунок 3 – Блок-схема алгоритма

Рисунок 4 – Блок-схема алгоритма

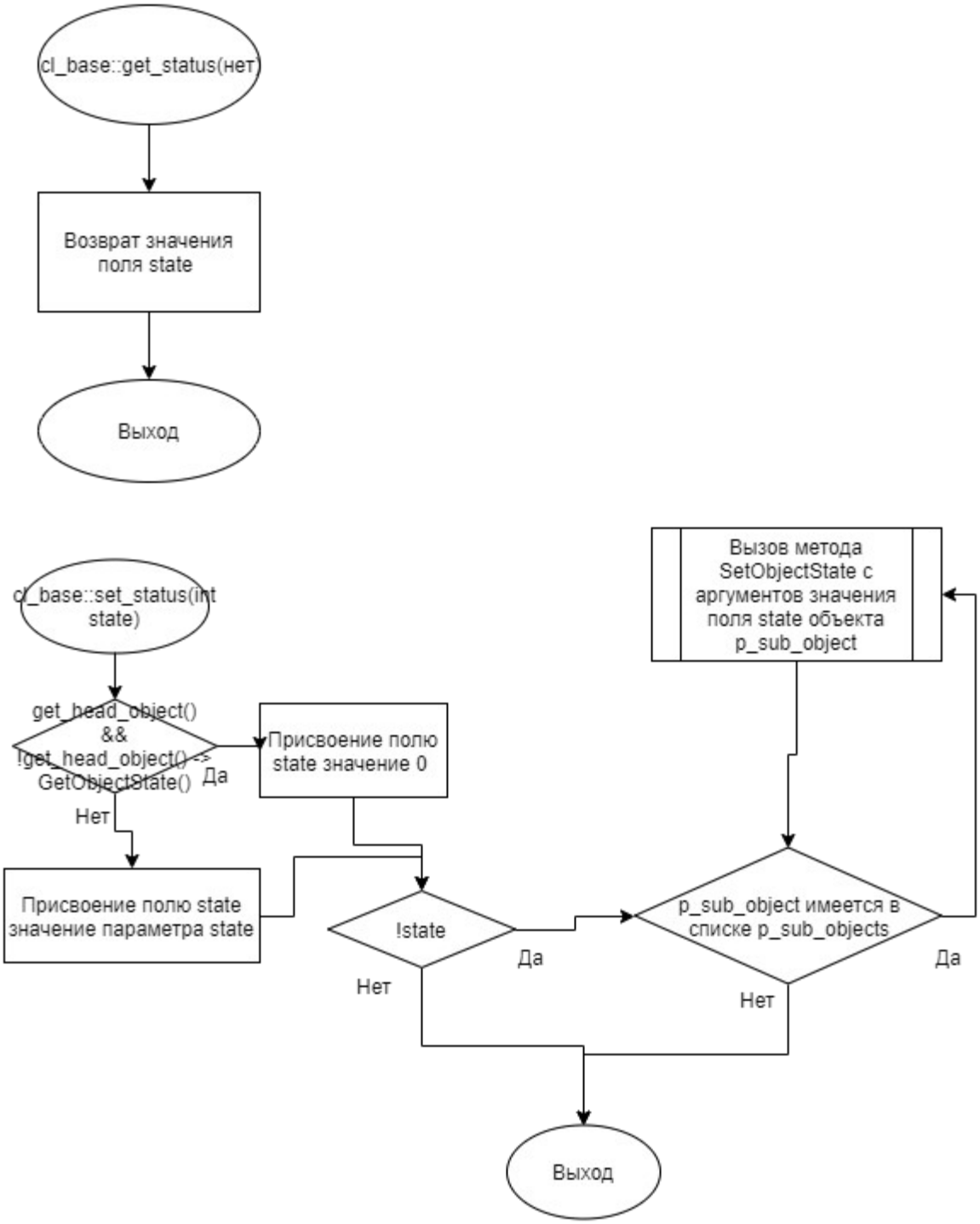

Рисунок 5 – Блок-схема алгоритма

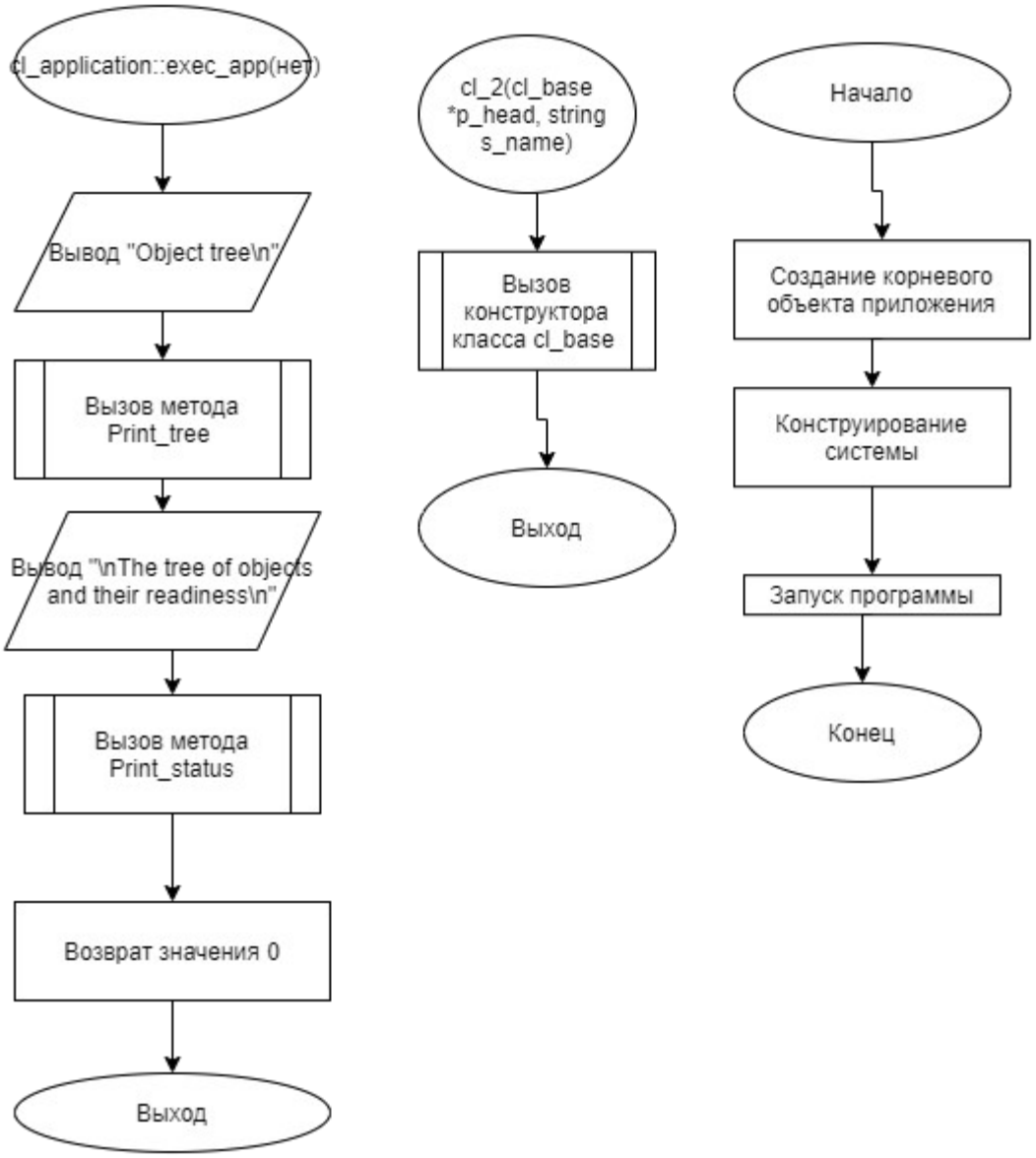

Рисунок 6 – Блок-схема алгоритма

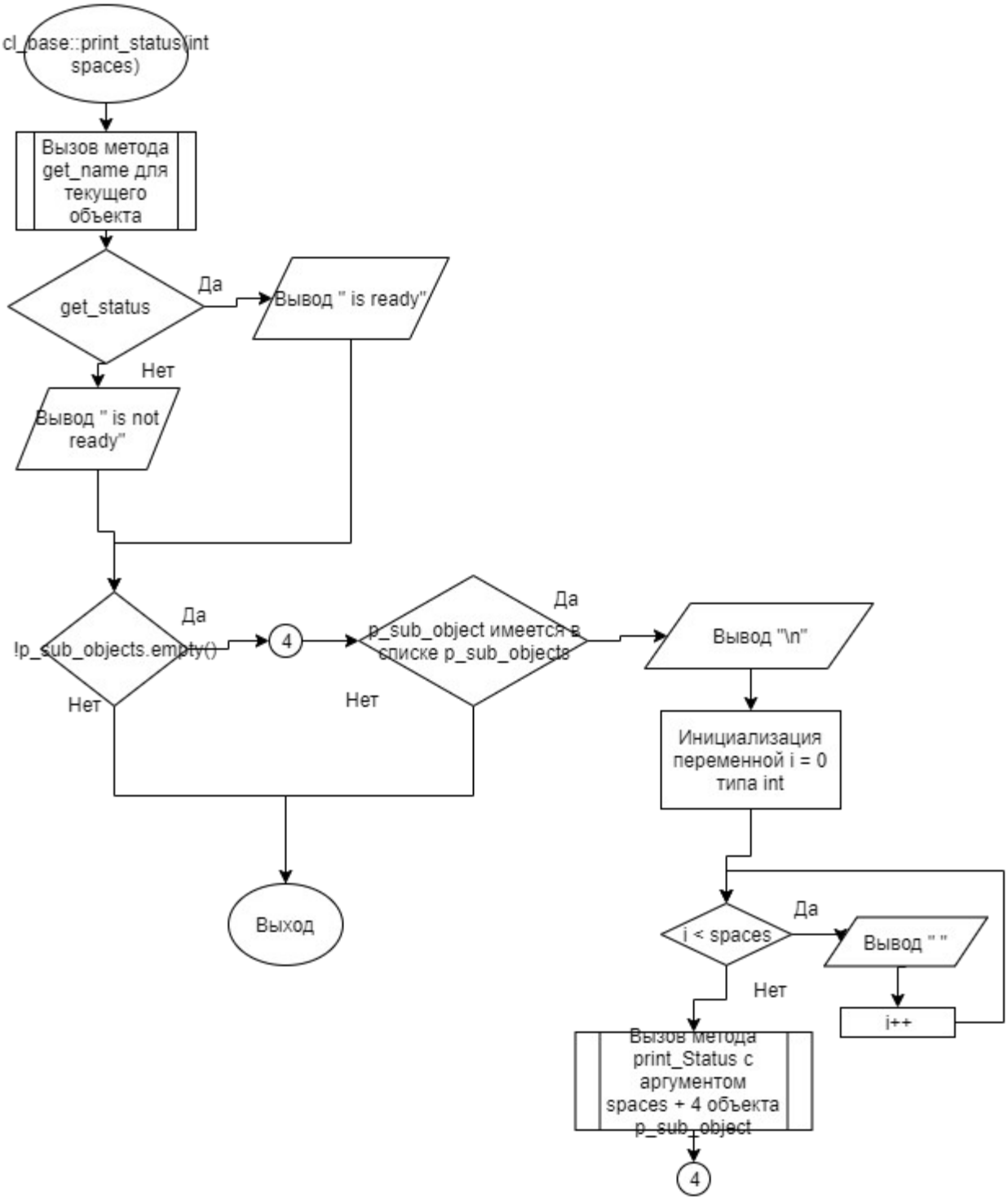

Рисунок 7 – Блок-схема алгоритма

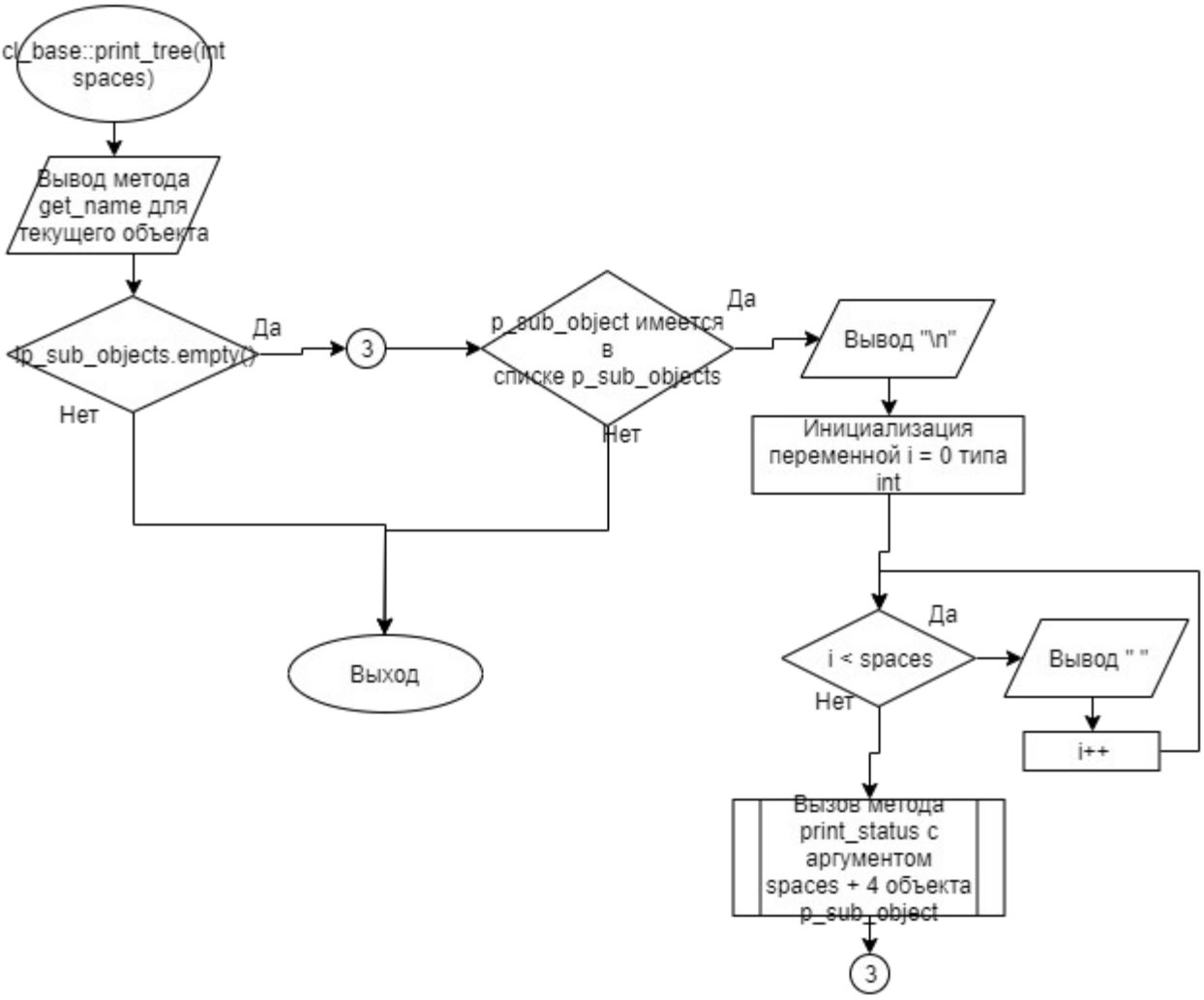

Рисунок 8 – Блок-схема алгоритма

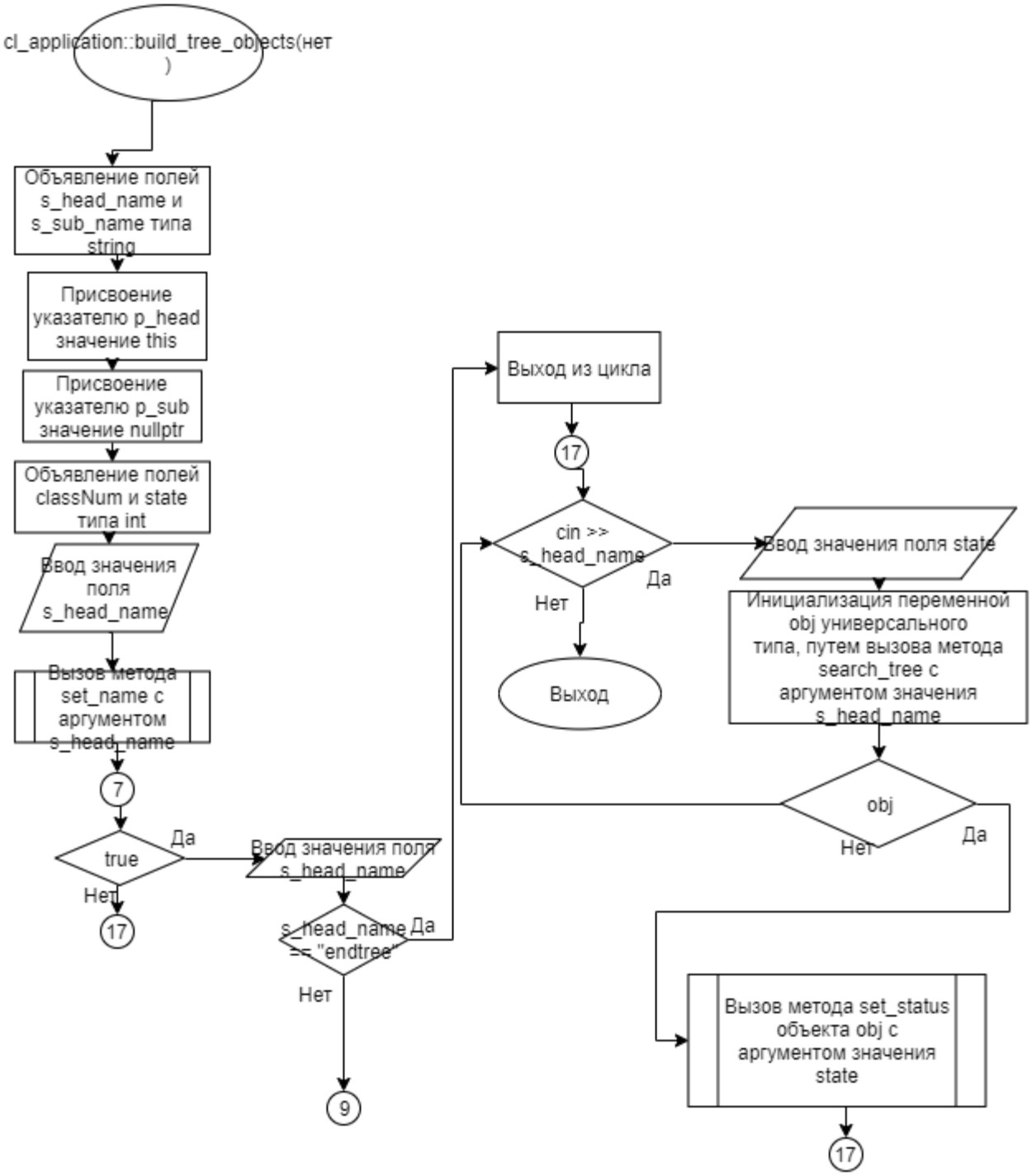

Рисунок 9 – Блок-схема алгоритма

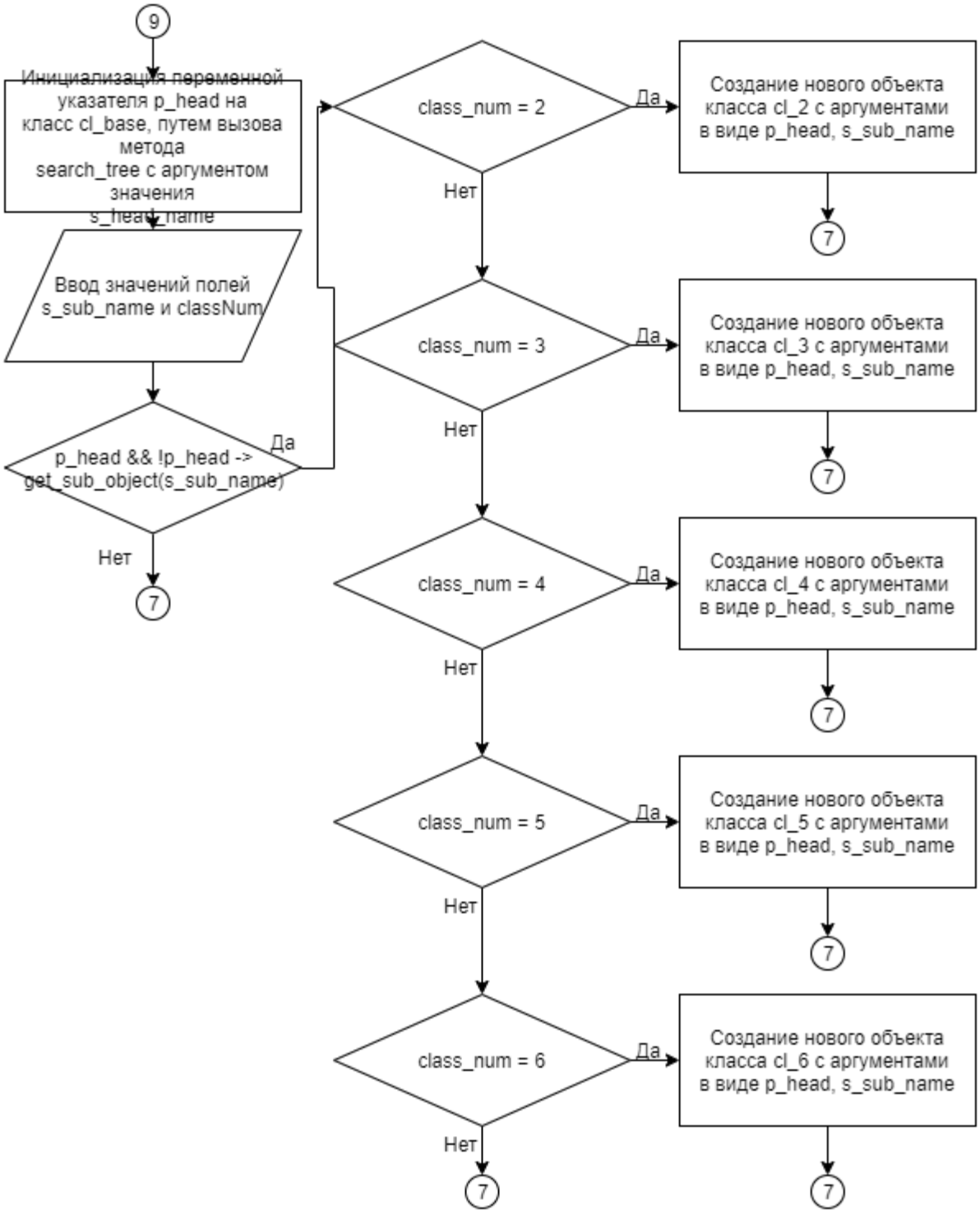

Рисунок 10 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"

cl_1::cl_1(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef CL_1__H
#define CL_1__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_1: public cl_base{
      public:
            cl_1(cl_base* p_head, string s_name);
};
#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"
#include <iostream>
using namespace std;

cl_2::cl_2(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.4 Файл cl_2.h

Листинг 4 – cl_2.h

```
#ifndef CL_2__H
#define CL_2__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_2: public cl_base{
      public:
            cl_2(cl_base* p_head, string s_name);
};
#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.6 Файл cl_3.h

Листинг 6 – cl_3.h

```
#ifndef CL_3__H
#define CL_3__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_3: public cl_base{
      public:
            cl_3(cl_base* p_head, string s_name);
};
#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.8 Файл cl_4.h

Листинг 8 – cl_4.h

```
#ifndef CL_4__H
#define CL_4__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_4: public cl_base{
      public:
            cl_4(cl_base* p_head, string s_name);
};


#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.10 Файл cl_5.h

Листинг 10 – cl_5.h

```
#ifndef CL_5__H
#define CL_5__H
#include "cl_base.h"
#include <iostream>
using namespace std;
```

```
class cl_5: public cl_base{
	public:
		cl_5(cl_base* p_head, string s_name);
};


#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.12 Файл cl_6.h

Листинг 12 – cl_6.h

```
#ifndef CL_6__H
#define CL_6__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_6: public cl_base{
	public:
		cl_6(cl_base* p_head, string s_name);
};


#endif
```

5.13 Файл cl_application.cpp

Листинг 13 – cl_application.cpp

```
#include "cl_application.h"
#include <iostream>
using namespace std;

cl_application::cl_application(cl_base* p_head_object): cl_base(p_head_object){}
```

```
void cl_application::build_tree_objects(){
	string s_head_name, s_sub_name;
	cl_base* p_head = this;
	cl_base* p_sub = nullptr;
	int class_num, state;

	cin>> s_head_name;
	this->set_name(s_head_name);

	while (true){
			cin>> s_head_name;
			if (s_head_name=="endtree"){
				break;
			}
		cl_base* p_head = search_tree(s_head_name);
		cin >> s_sub_name >> class_num;

		if(p_head && !p_head->get_sub_object(s_sub_name)){
			switch(class_num){
			case 2:
			{
				new cl_2(p_head, s_sub_name);
				break;
			}
			case 3:
			{
				new cl_3(p_head, s_sub_name);
				break;
			}
			case 4:
			{
				new cl_4(p_head, s_sub_name);
				break;
			}
			case 5:
			{
				new cl_5(p_head, s_sub_name);
				break;
			}
			case 6:
			{
				new cl_6(p_head, s_sub_name);
				break;
			}
		}
	}

}
	while (cin>> s_head_name) {
		cin >> state;
		auto obj = search_tree(s_head_name);
		if(obj){
			obj->set_status(state);
		}
	}
}
int cl_application::exec_app() {
```

```
	cout << "Object tree" << endl;
	this->print_tree();
	cout << "\nThe tree of objects and their readiness" << endl;
	this->print_status();
	return 0;
}
```

5.14 Файл cl_application.h

Листинг 14 – cl_application.h

```
#ifndef CL_APPLICATION__H
#define CL_APPLICATOIN__H
#include <iostream>
#include "cl_base.h"
#include "cl_1.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
using namespace std;

class cl_application: public cl_base{
	public:
		cl_application(cl_base* p_head_object);
		void build_tree_objects();
		int exec_app();
};
#endif
```

5.15 Файл cl_base.cpp

Листинг 15 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base* p_head_object, string s_name){
	this->p_head_object = p_head_object;
	this->s_name = s_name;
	if(p_head_object != nullptr) {
		p_head_object -> p_sub_objects.push_back(this); // добавление в состав
подчиненных головного объекта
	}
}

bool cl_base::set_name(string s_new_name){
	if(p_head_object != nullptr) {
		for (int i = 0; i < p_head_object->p_sub_objects.size(); i++){
```

```
                  if (p_head_object->p_sub_objects[i]->get_name()== get_name()) {
                        return false;
                  }
            }
      }
      this->s_name = s_new_name;
      return true;
}

string cl_base::get_name(){
      return this->s_name;
}

cl_base* cl_base::get_head_object(){
      return this->p_head_object;
}

void cl_base:: print_tree(int spaces ){
      cout << this->get_name();
      if(!p_sub_objects.empty()){
            for(auto child : p_sub_objects){
                  cout << endl;
                  for(int i = 0; i < spaces; i++)
                        cout << " ";
                  child->print_tree(spaces+4);
            }
      }
}

cl_base::~cl_base(){
      for (int i=0 ; i < this->p_sub_objects.size(); i++) {
                  delete p_sub_objects[i];
            }
}

cl_base* cl_base::get_sub_object(string S_name){
      for (auto sub: p_sub_objects) {
                  if (sub->get_name() == S_name) {
                        return sub;
                  }
            }
      return nullptr;
}



int cl_base::obj_count(string s_name){
      int count = 0;
      if(this->get_name() == s_name){
            count++;
      }
      for (auto child : p_sub_objects){
            count += child->obj_count(s_name);
      }
      return count;
}
```

```
cl_base* cl_base::search_object(string s_name){
	if (this-> obj_count(s_name)!=1 ){
		return nullptr;
	}
	if (this -> get_name() == s_name){
		return this;
	}
	for (auto p_sub_object: p_sub_objects){
		cl_base* p_found = p_sub_object->search_object(s_name);
		if(p_found != nullptr){
			return p_found;
		}
	}
	return nullptr;
}

cl_base* cl_base::search_branch(string s_name){
	if (this-> obj_count(s_name) !=1 ){
		return nullptr;
	}
	return search_object(s_name);
}

cl_base* cl_base::search_tree(string s_name){
	auto obj = this;
	while(obj->get_head_object()){
		obj = obj->get_head_object();
	}
	if(obj_count(s_name) != 1){
		return nullptr;
	}
	return search_object(s_name);
}


void cl_base::print_status(int spaces){
	cout << this->get_name();
	cout << (get_status() ? " is ready" : " is not ready");
	if(!p_sub_objects.empty()){
		for(auto sub: p_sub_objects){
			cout << endl;
			for (int i =0; i < spaces; i++)
				cout<< " ";
			sub->print_status(spaces+4);
		}
	}
}



void cl_base::set_status(int state){
	if(get_head_object() && !get_head_object()->get_status()) {
		this->state=0;
	}
	else {
		this->state = state;
	}
```

```
      if(!state){
            for (auto sub: p_sub_objects){
                  sub->set_status(state);
            }

      }
}

bool cl_base::get_status() const{
      return state;
}
```

5.16 Файл cl_base.h

Листинг 16 – cl_base.h

```
#ifndef CL_BASE__H
#define CL_BASE__H
#include <vector>
#include <string>
#include <iostream>
#include <queue>
using namespace std;

class cl_base{

      string s_name;
      cl_base* p_head_object;
      vector <cl_base*> p_sub_objects;
      int state = 0;

      public:
      cl_base(cl_base* p_head_object, string s_name= "Base_object");
      bool set_name(string s_new_name);
      string get_name();
      cl_base* get_head_object();
      void print_tree(int spaces=4);
      ~cl_base();
      cl_base* get_sub_object(string S_name);
      int obj_count(string s_name);
      cl_base* search_object(string s_name);
      cl_base* search_branch(string s_name);
      cl_base* search_tree(string s_name);
      void print_status(int spaces=4);
      void set_status(int state);
      bool get_status()const;
};
#endif
```

5.17 Файл main.cpp

Листинг 17 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "cl_application.h"
using namespace std;

int main()
{
      cl_application ob_cl_application(nullptr);
      ob_cl_application.build_tree_objects();
      return ob_cl_application.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 14.

Таблица 14 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
object_02 object_05 5
object_01 object_07 2
endtree
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
	Object tree
	app_root
	    object_01
	        object_07
	    object_02
	        object_04
	        object_05
	The tree of objects and their readiness
	app_root is ready
	    object_01 is ready
	          object_07 is not ready
	    object_02 is ready
	        object_04 is ready
	          object_05 is not ready
		Object tree
		app_root
		    object_01
		        object_07
		    object_02
		        object_04
		        object_05
		The tree of objects and their readiness
		app_root is ready
		    object_01 is ready
		          object_07 is not ready
		    object_02 is ready
		        object_04 is ready
		          object_05 is not ready

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).